Implementation Phase

Third phase: Implementation

- In this phase, the algorithms described in the design phase are converted into programming code after choosing the appropriate programming language.
- Each programmer receives a module (or a part of it) details in terms of a flowchart or a pseudo code, then using the appropriate programming language, he implements this part.

How to choose the appropriate programming language:

- Each application may have more than one appropriate programming language.

Third phase: Implementation

- The programmer may choose any of them according to his experience.
- To be a good programmer, you must:

1- learn the theoretical background of the language.

2- apply this background practically as only through practice and try and error you can improve your skills. The more time you spend on practice, the more the experience you get.

3- You can use the experience and help from other programmers to add to your experience.

Third phase: Implementation

Application	Appropriate PL
Scientific or mathematical	FORTRAN, C++, Pascal, Basic
Graphics	Java, Visual Basic, C++
Database	Access package + visual Basic or SQL server, Oracle
Manipulating Hardware	C++ , Assembly
Web Design	HTML + Java script, VB script, PHP, XML
Artificial Intelligence	Lisp, Prolog
Hardware Programming	VHDL, Verilog

Programming Considerations

Program names:

- Choose the appropriate program names such as names of : variables, functions, arrays, classes,…
- Not too small, not too long and not expressive.

Example: Choose the name of variable that will hold the radius of a circle.

- Radius_of_circle — too long
- R — too small
- RA — not expressive
- Rad or Radius

Programming Considerations

Program Structure:

- The program should be written in a n organized from to facilitate reading it in a further step. Examples:
- Leave a blank line between functions description.
- Avoid writing two or more statements in the same line (as in C++).
- Write the first letter in function name, arrays, classes and data structures in capital letter to characterize them.
- Put all declaration after function header to easily locate them.

Programming Considerations

- Add comments to the difficult parts to be able to memorize their function later.
- Put the body of if, if-else-if, for, switch in { }.
- Put { } in separate lines with no statements in them.
- Intend the body of if, if-else-if, for, switch as in

```
If(x>y)
{
   statements
}
Else
{
   statements
}
```

Programming Considerations

Program control structure:

- **Use the appropriate control structure (if, for, switch, while) that will achieve the required function with less number of lines and less complexity.**

Example:

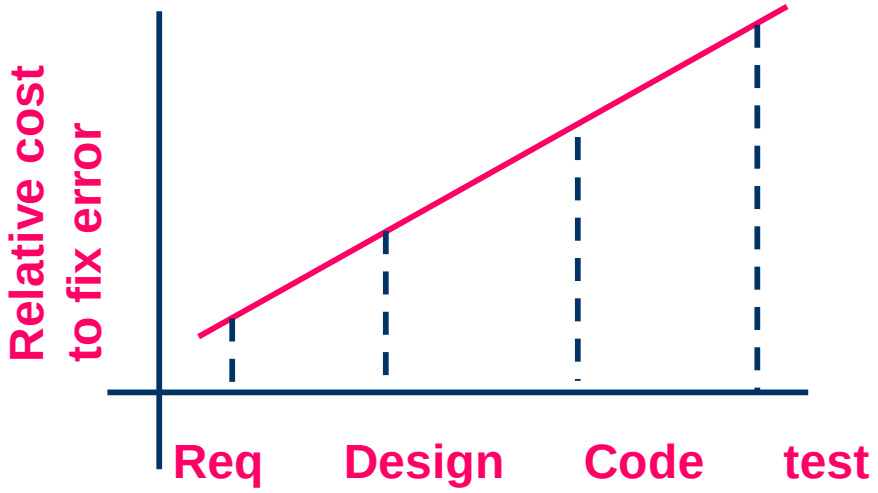
If(c==1) statements
Else if(c==2) statements
Else if(c==3) statements
Else statements

Switch (c):
Case 1: statements
Case 2: statements
Case 3: statements
Default: statements

Test Phase

Fourth Phase: Test

- Testing the SW is very important specially in SWs that are used to control sensitive devices such as: autopilot in airplanes, or nuclear power plant.
- At each previous phase, there is an inherent test phase.
- The earlier we find the error the less the cost of its fixing.



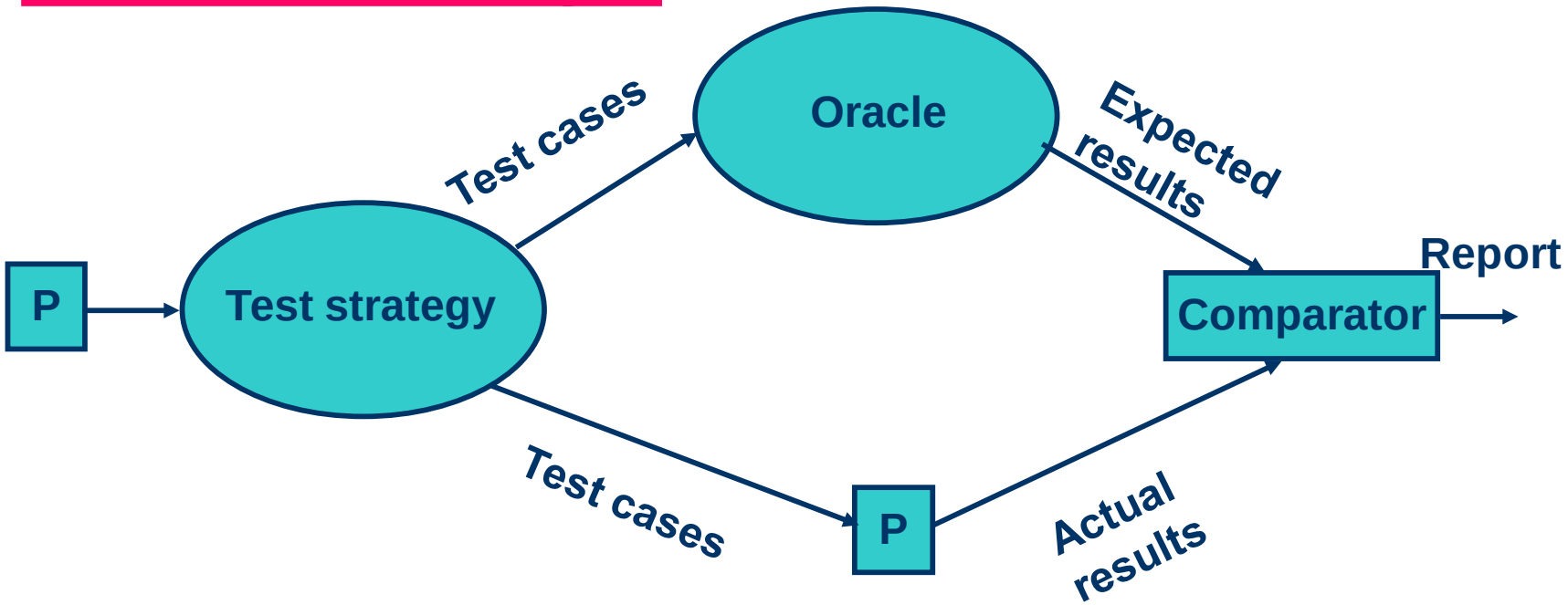
Fourth Phase: Test

- Test of SW shows only the presence of errors, not their absence.
- **Error:** A human action that produces an incorrect result.
- **Fault:** The consequence of an error in SW lines.
- **Failure:** The result of fault during the program execution.
- When we test the SW, we observe failure. These failures are caused by faults, which in turn are the result of human error.

Fourth Phase: Test

- A failure may be caused by more than one fault and a fault may cause different failures.

Test Process steps:



Test Process steps

- P: object to be tested (program, design document, module,…)
- Test strategy: decide show to test the object:
- Test cases: they are the conditions applied to the tested program to test whether it performs its function in a proper way. They depends on the type of the SW. They could be constant input data values, or test of a specific task such as cut, paste, print,…
- Oracle: a way to determine the expected output for the test cases.
- Test cases are applied to P to produce the actual results.

Test Process steps:

- **Compare: compares the actual output to the expected output.**
- **Make a report of the result of comparison.**

Test adequacy criteria:

- **The percentage of program lines to be tested.**
- **Test adequacy criteria = 100% all program lines**
- **Test adequacy criteria = 75% 75% of program lines**
- **Test adequacy criteria = 50% half program lines**
- **In critical programs, we need to have high Test adequacy criteria.**

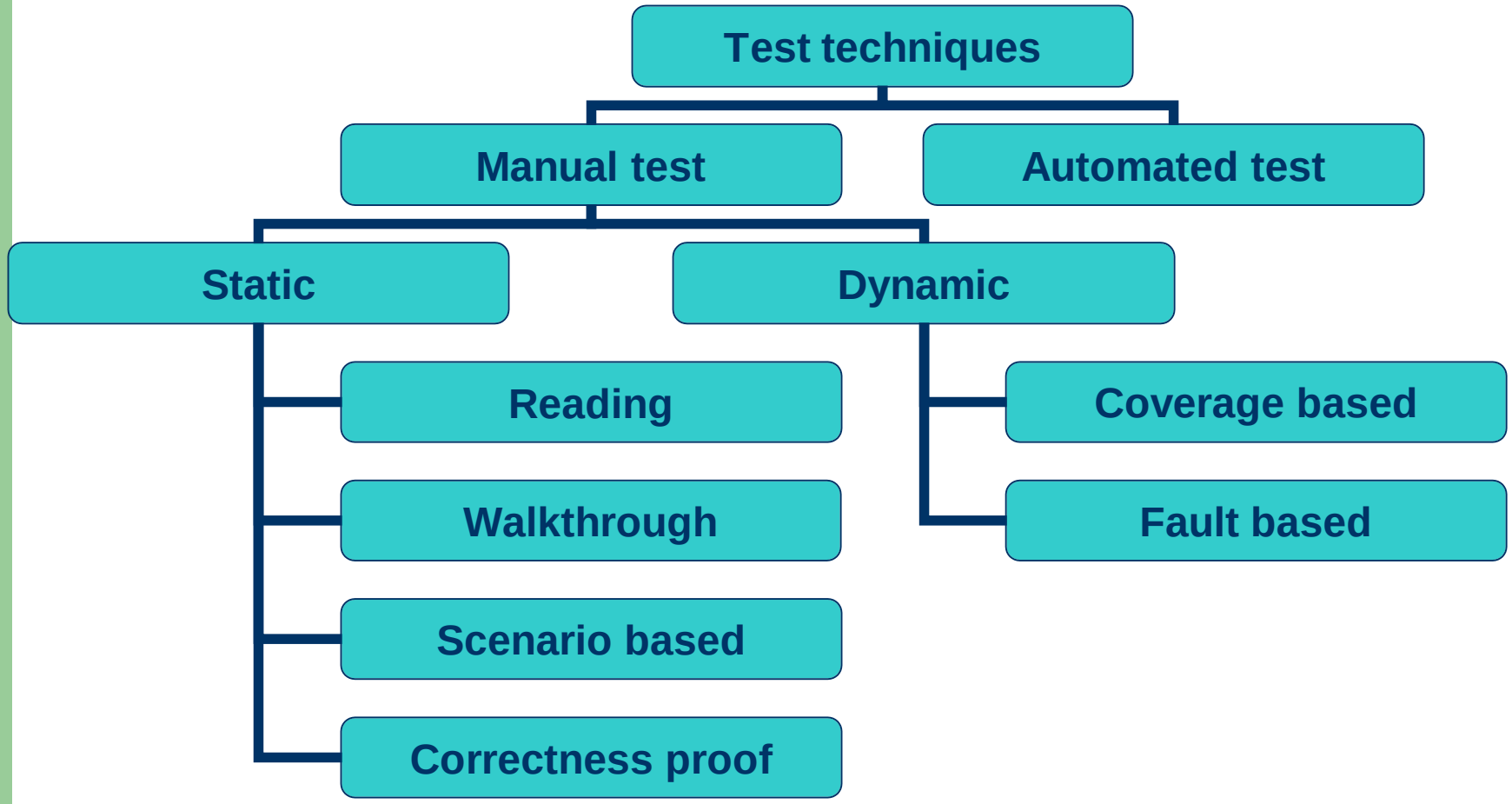
Test adequacy criteria

Example:

```
Cin>>x,y;
If(x>y)
Cout<<"X is geater than Y";
Else
Cout<<"X is geater than Y";
```

- **Test adequacy = 100% test cases={x=3,y=5}, {6,2}**
- **Test Adequacy = 50% test cases= {3,5}**

Test techniques



Test techniques

Automated test:

- **Uses an intelligent program that receives the tested object and test cases, then it tests the object generating a report. Not available.**

Manual test:

- **Human test of the object.**

a) Static test; we test the program without making executable (.exe) file of the program. Made by reading the program (or any tested object).

Static test techniques

i) Reading (no test cases are used):

- **The program is read line by line to find errors by one tester.**
- **The complier does the same task of reading test instead of human.**
- **Suitable for small size programs or for documents.**

ii) Walkthrough: (no test cases are used):

- **The program is read line by line to find errors but instead by a group of tester, one reads and they walkthrough the lines finding errors and taking notes.**

Static test techniques

iii) Scenario-based test: (test cases are used)

- **We follow the program lines imagining the scenario of program execution after entering the test cases .**

iv) Correctness proof: (test cases are used)

- **We have an expected output for the test cases.**
- **We follow the program lines after entering the test cases until we get the actual result.**
- **We compare both results and find the place of error.**

Dynamic test techniques

- We test the program after making an executable (.exe) file of the program.

1) Coverage-based test:

- Chooses the test cases that cover a percentage of the program lines.
- Needs a flowchart to understand the structure of the program and to determine the test cases.
- The program is executed number of times with different test cases that cover the required %.

Dynamic test techniques

2) Fault-based test

- This type inserts an intentional error in the program lines, which its result is expected.
- The actual result of executing the program with the inserted errors is compared to the expected result with error.
- If they match, then there is no error except that inserted one.
- If not, then there is an error(s) elsewhere in the program that must be discovered by any means

Dynamic test techniques

- **Example:**

Without intentional error:	After inserting error;
cin>>x,z;	*cin>>x,z;*
z= x+y;	*z= x-y;*
cout<<z;	*cout<<z;*

- Test cases (3,5). Expected result with error = -2.
- The actual result will be garbage as y want not initialized or set to a value.
- The error is in line 1 not in line 2 (z should be replaced by y).

Maintenance Phase

Fifth Phase: Maintenance

- It is the process that keeps the SW running at a proper level that satisfies the user.
- It includes the following activities:

1- Repairing faults found

2- Adapting to environmental changes

3- Adding or modifying functionality to the SW

Types of maintence activities:

1- Correcting

2- Adapting

3- Perfective

4- Preventive

1- Corrective Maintenance

- It is concerned with correcting any faults found by the SW users after delivery and at run time.

- These faults were not discovered at the SW company at the test phase due to many factors such as:
  - Low test adequacy.
  - Short time assigned to test phase.
  - Complexity of the SW under which the whole functions can not be tested.

2- Adaptive Maintenance

- It is responsible for making any changes in the SW that makes it work under different environment other than the one which is was designed to work under such as:
  - Different Operating system
  - Different hardware configuration
  - Different number of users
  - Different backgrounds or fonts

- Thus, the functions of the system are not changed, only some variables in these functions are changed.

3- Perfective Maintenance

- It is responsible for adding new functions to the SW which the user requires to improves the functionality of the SW.
- So, the old version of the SW is working with no faults, yet with less functions than required.

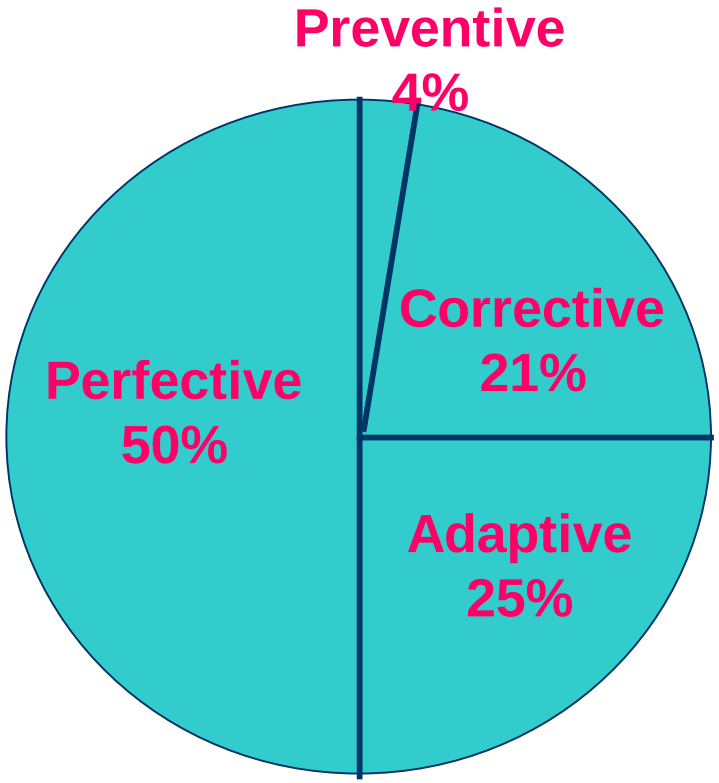
4- Preventive Maintenance

- It is responsible for maintaining the SW documents after making any of the three previous activities.
- This is to keep the documents up-to-date with the working version of the SW.
- This will help performing any other activities on the SW which will require looking at the details of the design, code documents of the current SW version.
- If this step is not performed, the documents will be not compatible with the working version of SW and locating the place to make the maintenance activity will be very hard problem to the maintenance engineer.

Maintenance activities distribution

- We notice from the distribution that:

1- The highest % is for the perfective for many reasons such as: the requirement phase was not performed in the right way to capture all the requirements of the user, and the demanding for change natural of the users which keeps them in continuous demand for new functions.



Maintenance activities distribution

2- The next high % is for adaptive which is due to the changing nature of the OS and HW configurations, and due to the inflexibility of the SW itself as it did give the user the ability to make these changes himself.

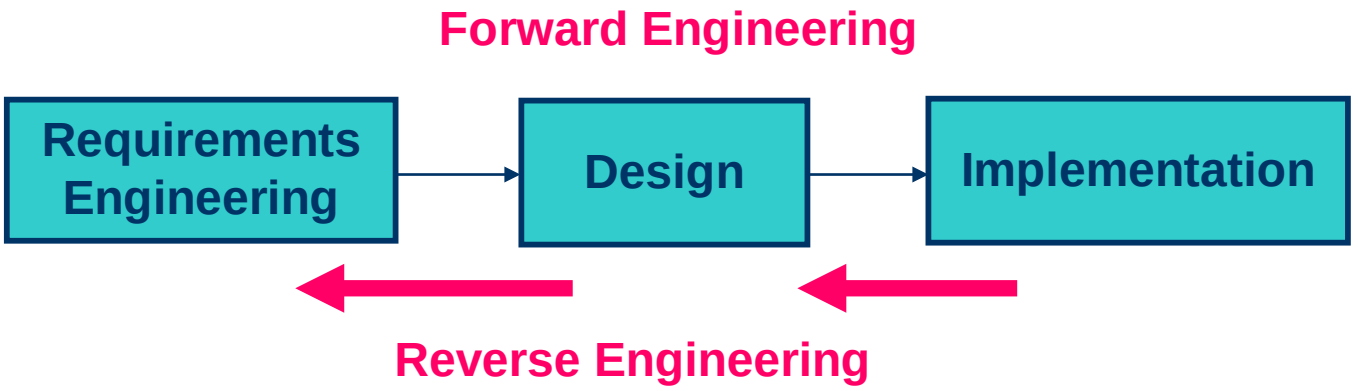
3- The corrective activity is high specially in the complicated SW and the little time given to test phase.

Problems facing the maintenance engineer

1) Program is not clear either in its name or structure.
2) It is hard for him to understand the problem domain.
3) If the design was bad, it will be hard for him to add new functions (perfective) or to change some parameters to adapt to the environment changes.
4) The documents may be lost or not up-to-date.

- What are the consideration to be made to solve these problems??????????

Reverse SW engineering



- It is one of the maintenance engineer possible activities.
- Used if the SW documents are missing.
- It is the science of reproducing the SW documents from the currently working version of the SW.

SW Restructuring

- If the SW structure is poor, the maintenance engineer can restructure the current SW.
- This is made without changing the services (functions) of the SW.
- It only organize these functions in a proper way, externally or internally.
- Externally, by regrouping them in different menus.
- Internally, by changing the control structures or program names.

Maintenance process steps

1) Collect information about the activities required.
2) Organize them in a report specifying the type of each activity, what is required, the estimated cost.
3) This report is delivered to the manger who will determine the priority of each step according to the company's condition and the client's approval.
4) The chosen activities are scheduled to be performed specifying the time of each sub-activity and the role of each worker.
5) The SW is tested and then delivered to the users.

SW Cost Estimation

- The cost of producing the SW = Requirement cost + Design Cost + Implementation Cost + Test Cost
- A traditional way is to compute the cost as = number of workers x number of working hours x hourly salary.
- But, in many cases, the team contains different persons each with different experience, thus the hourly salary is not equal for them.
- One other issue, the SW complexity is another factor to be considered. Complicated SW costs more than simple SW.

SW Cost Estimation

- After making intensive research, several equations that computes the SW cost were developed, among them the following equation:

$$E = 2.7(KOLC)^{1.05}$$

KLOC = Kilo Line Of Code

- This equation relates the to the effort done to make the SW in terms of lines produced.
- Thus to minimize the SW cost in terms of number of lines, we can use experienced programmers that could produce the same functions in less number of lines. We can also reuse parts from other programs to minimize the new number of lines.

SW Quality

- The quality of the SW can be measured from different points of view.
- Each ISO (International Standards Organization) has its own standards. (ISO 9001)
- Examples of SW quality measures:

1) Correctness:

- Does the SW perform its tasks properly? Are these the requirements that the users demanded?

2) Reliability:

- Does the SW performs its tasks accurately each time?

SW Quality

3) Flexibility:

- Easy to make adaptation or perfection maintenance to it.

4) Portability:

- The ability to work under different environments.

5) Testability:

- Easy to be tested.

6) Interoperability:

- Able to work with other programs at the same time. Or the output of this SW can be used by other SW.

SW Quality

7) Reusability:

- The extent to which its components can be reused in other SW building.

8) Usability:

- The effort required to learn, operate, prepare inputs and interpret outputs of the SW.

9) Maintainability:

- The effort required to locate and fix errors in the SW.